Simulation Assignment

Overview

Study the impact of a scheduling algorithm on the average turnaround time of concurrent processes.

A simulation mimics the execution of n different processes under a scheduling algorithm. The simulation maintains a table that reflects the current state of the system:

Process	**Active**	**Arrival Time**	**Total CPU Time**	**Remaining CPU Time**	**Turnaround Time**
P_i	0/1	A_i	T_i	R_i	TT_i

The table initializes as follows:

1. The field "active" indicates whether the process is currently competing for the CPU. The value becomes 1 at the time of process arrival and 0 at the time of process termination. Initially, the value is set to 1 for all processes with arrival time $A_i = 0$.
2. Each A_i is an integer chosen randomly from a uniform distribution between 0 and some value k, where k is a simulation parameter.
3. Each T_i is an integer chosen randomly from a normal (Gaussian) distribution with an average d and a standard deviation v, where d and v are simulation parameters.
4. Each R_i is initialized to T_i, since prior to execution, the remaining time is equal to the total CPU time required.

The simulation maintains the current time, t, which is initialized to 0 and is incremented after each simulation step. Each simulation step then consists of the following actions within the repeat loop:

```
repeat
   increment t
   if there is at least one active process
      choose from the active processes a process Pi to run next,
         according to the scheduling algorithm.
      decrement Ri (Pi has accumulated 1 CPU time unit)
      if Ri == 0 (process i has terminated)
         set active flag of Pi = 0
         TTi = t - Ai
   end if
until Ri == 0 for all n processes (all processes have terminated)

compute the average turnaround time, ATT, by averaging all values TTi
```

Design

1. Implement the above simulation in Java for the First Come First Serve scheduling algorithm.

2. Choose a value for n.

3. Choose a value for k, which is the time interval during which processes may arrive. Ex: k = 100.

4. Using a random number generator, derive n arrival times, A_i, for all processes, distributed uniformly within the interval [0 : k]. Note: the java.util.Random class provides a random number generator. Create an instance of this class and use the nextDouble() method to get a random number from the range of 0.0 to 1.0. Multiply the random number by k to generate an arrival time A_i.

5. Choose an average total CPU time, d, and a standard deviation, v, and derive n total CPU times, T_i, using a normal (Gaussian) distribution. Note: the java.util.Random class instance from number 3 above also has a nextGaussian() method which generates a Gaussian distributed random number from the range of 0.0 to 1.0. Multiply the Gaussian distributed random number by the standard deviation v and add the average total CPU time d to generate a total CPU time T_i.

6. The values d and v should be selected relative to k. For simplicity, v could be just be a fixed percentage of d. The value k/n represents the frequency of process arrivals. When d is much smaller than k/n, then most processes run in isolation, without having to compete with other processes for the CPU. On the other hand, when d is much larger than k/n, then many processes will overlap in time and compete for the CPU. Choose a value for d that is much larger than k/n.

7. One or more classes implement the simulator and one or more classes implement the First Come First Serve scheduling algorithm.

8. Create a driver class and make the name of the driver class **Assignment1** containing only one method: `public static void main(String args[])`. The main method receives, via the command line arguments, the values for n, k, d, and v, in that order. The main method creates the simulator object, the scheduling algorithm object, and initiates the execution of the simulator.

   I compile and run your program via the command line using the Java JDK. Therefore, the command I type to execute your program is **java Assignment1**. I'll test your program with my own values for n, k, d, and v.

   For example, if n = 100, k = 1000, d = 40, and v = 8 then the command to run the program is:

   ```
   java Assignment1 100 1000 40 8
   ```

9. You must declare public each class you create which means you define each class in its own file.

10. You must declare private all the data members in every class you create.

11. You **do not** need and **cannot** use **extends** in this assignment.

12. **Tip:** Make your program as modular as possible, not placing all your code in one .java file. You can create as many classes as you need in addition to the classes described above. Methods being reasonably small follow the guidance that "A function does one thing and does it well." You will lose a lot of points for code readability if you don't make your program as modular as possible. But, do not go overboard on creating classes and methods. Your common sense guides your creation of classes and methods.

13. Do **NOT** use your own packages in your program. If you see the keyword **package** on the top line of any of your .java files then you created a package. Create every .java file in the **src** folder of your Eclipse project, if you're using Eclipse.

14. Do **NOT** use any graphical user interface code in your program!

15. Do **NOT** type any comments in your program. If you do a good job of programming by following the advice in number 12 above then it will be easy for me to determine the task of your code.

Grading Criteria

The total assignment is worth 20 points, broken down as follows:

1. If your code does not implement the task described in this assignment then the grade for the assignment is zero.
2. If your program does not compile successfully then the grade for the assignment is zero.
3. If your program produces runtime errors which prevents the grader from determining if your code works properly then the grade for the assignment is zero.

If the program compiles successfully and executes without significant runtime errors then the grade computes as follows:
Followed proper submission instructions, 4 points:

1. Was the file submitted a zip file.
2. The zip file has the correct filename.
3. The contents of the zip file are in the correct format.
4. The keyword **package** does not appear at the top of any of the .java files.

Code implementation and Program execution, 12 points:

- The driver file has the correct filename, **Assignment1.java** and contains only the method **main** performing the exact tasks as described in the assignment description.
- The code performs all the tasks as described in the assignment description.
- The code is free from logical errors.
- Program output, the program produces the correct results for the input.

Code readability, 4 points:

- Good variable, method, and class names.
- Variables, classes, and methods that have a single small purpose.
- Consistent indentation and formatting style.
- Reduction of the nesting level in code.

Late submission penalty: assignments submitted after the due date are subjected to a 2 point deduction for each day late.

Late submission policy: you **CAN** submit your assignment early, before the due date. You are given plenty of time to complete the assignment well before the due date. Therefore, I do **NOT** accept any reason for not counting late points if you decide to wait until the due date (and the last possible moment) to submit your assignment and something happens to cause you to submit your assignment late.

Submission Instructions

Go to the folder containing the .java files of your assignment and select all (and **ONLY**) the .java files which you created for the assignment in order to place them in a Zip file. The file can **NOT** be a **7z** or **rar** file! Then, follow the directions below for creating a zip file depending on the operating system running on the computer containing your assignment's .java files.

Creating a Zip file in Microsoft Windows (any version):

1. Right-click any of the selected .java files to display a pop-up menu.
2. Click on **Send to**.
3. Click on **Compressed (zipped) Folder**.
4. Rename your Zip file as described below.
5. Follow the directions below to submit your assignment.

Creating a Zip file in Mac OS X:

1. Click **File** on the menu bar.
2. Click on **Compress ? Items** where ? is the number of .java files you selected.
3. Mac OS X creates the file **Archive.zip**.
4. Rename **Archive** as described below.
5. Follow the directions below to submit your assignment.

Save the Zip file with the filename having the following format:

your last name,
followed by an underscore _,
followed by your first name,
followed by an underscore _,
followed by the word **Assignment1**.

For example, if your name is John Doe then the filename would be: **Doe_John_Assignment1**

Once you submit your assignment you will not be able to resubmit it!
Make absolutely sure the assignment you want to submit is the assignment you want graded.
There will be **NO** exceptions to this rule!

You will submit your Zip file via your CUNY Blackboard account.
The only accepted submission method!

Follow these instructions:

Log onto your CUNY BlackBoard account.
Click on the CSCI 340 course link in the list of courses you're taking this semester.
Click on **Content** in the green area on the left side of the webpage.
You will see the **Assignment 1 – Simulation Assignment**.
Click on the assignment.
Upload your Zip file and then click the submit button to submit your assignment.

Due Date: Submit this assignment by Tuesday, April 28, 2020.